Theory of Automata and Languages

Turing Machines and Turing-recognizable Languages - 1

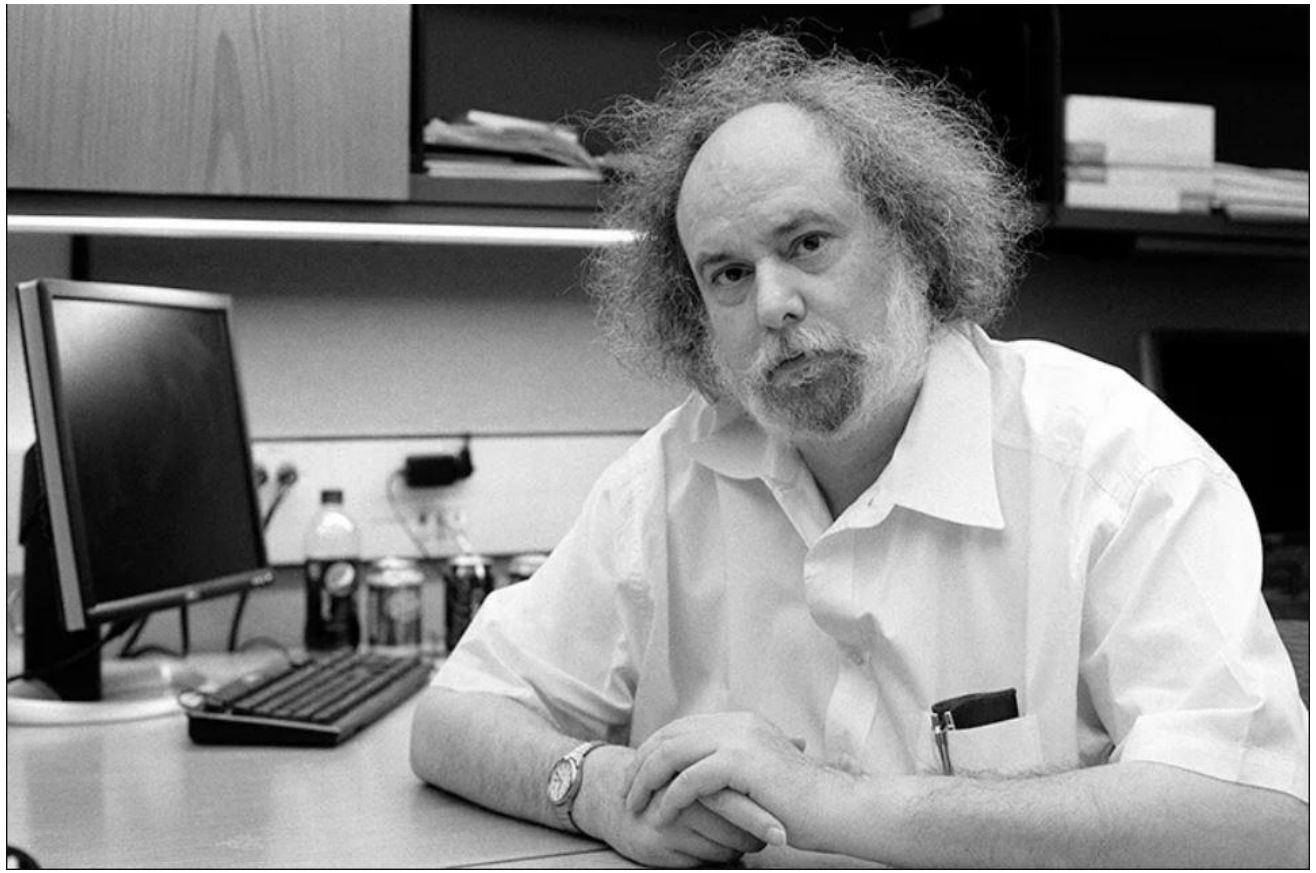
Fall 2024

Sharif University of Technology

Mehran Moeini Jam



In Loving Memory of
E. Allen Emerson
1954 - 2024



Overview

- Deterministic Turing Machine - as an acceptor
- String and Language Recognition
- Multi-tape Turing Machines
- Non-deterministic Turing Machine
- Function Problems vs. Decision Problems
- The Definition of Algorithm
- The Church-Turing Thesis

Turing Machine

- The Turing Machine model was first proposed by Alan Turing in 1936, and

  further popularized by his mentor, Alonzo Church, who named it after Turing.



Alan M. Turing



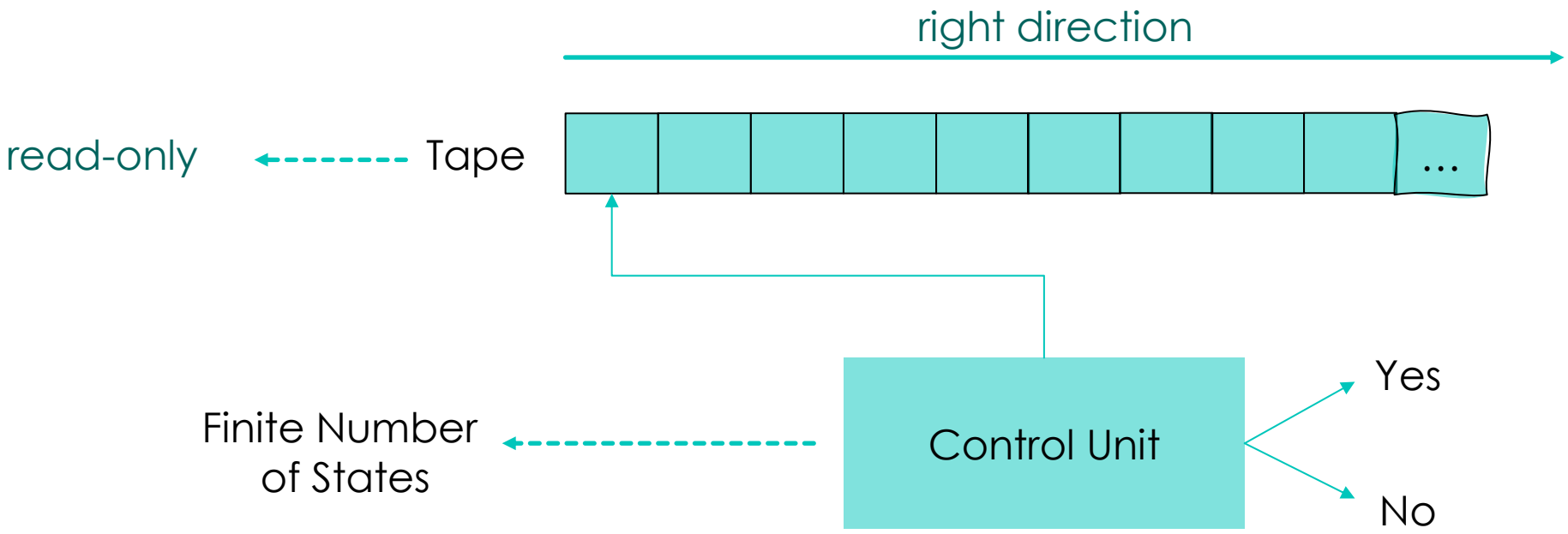
Alonzo Church

Turing Machine

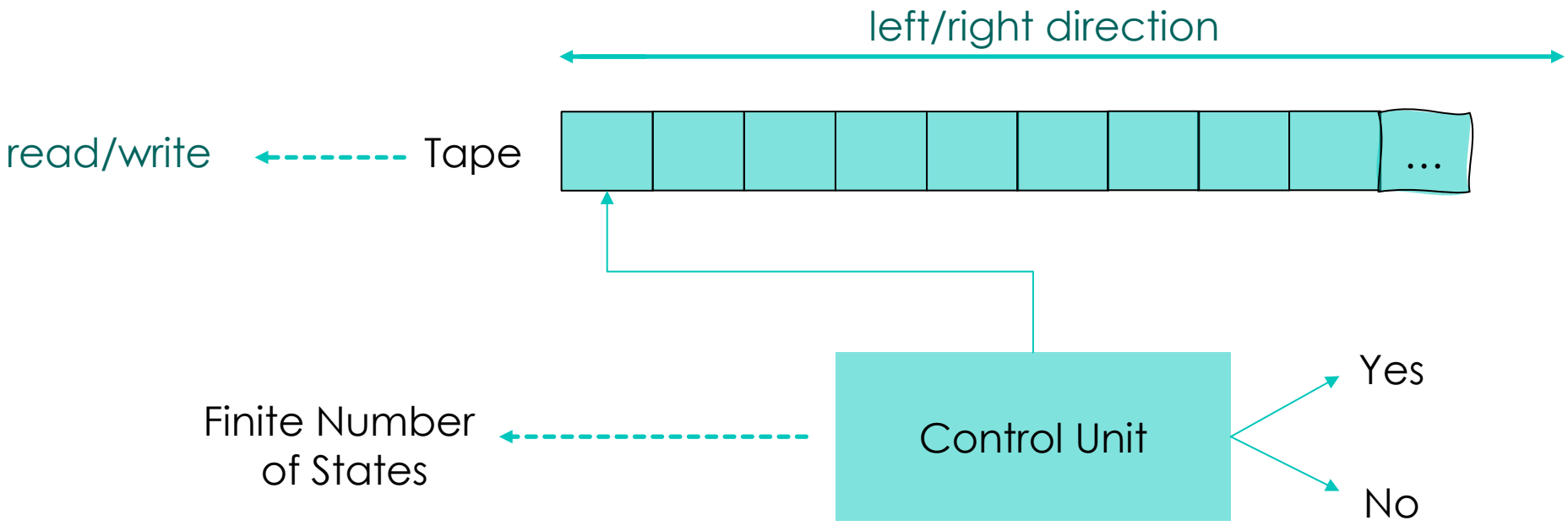
- Commercial computers are all limited versions of Turing machine. Nonetheless, even a Turing machine cannot solve certain problems.

- In a very real sense, these problems are beyond the theoretical limits of computation, as depicted by the Church-Turing thesis, that we will see later.

Finite[-State] Automaton (as an acceptor)



Turing Machine (as an acceptor)



Turing Machine (as an acceptor)

- The operations read, write, move, and change state are the basic (or atomic) operations of the Turing machine. Complex operations are achieved by chaining together large numbers of these simple basic actions.

- On the other side, commercially available computers are hard-wired to perform basic operations considerably more sophisticated than those of a Turing machine — add, multiply, decrement, store-at-address, branch, and so forth.

Turing Machine (as an acceptor)

- It is a remarkable fact, however, that despite the simplicity of Turing machines, they are capable of computing anything that any computer on the market can compute.

- Indeed, because they are abstract machines, with unlimited memory, they are capable of computations that no actual computer could perform in practice.

Deterministic Turing Machines (DTM)

A Deterministic Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

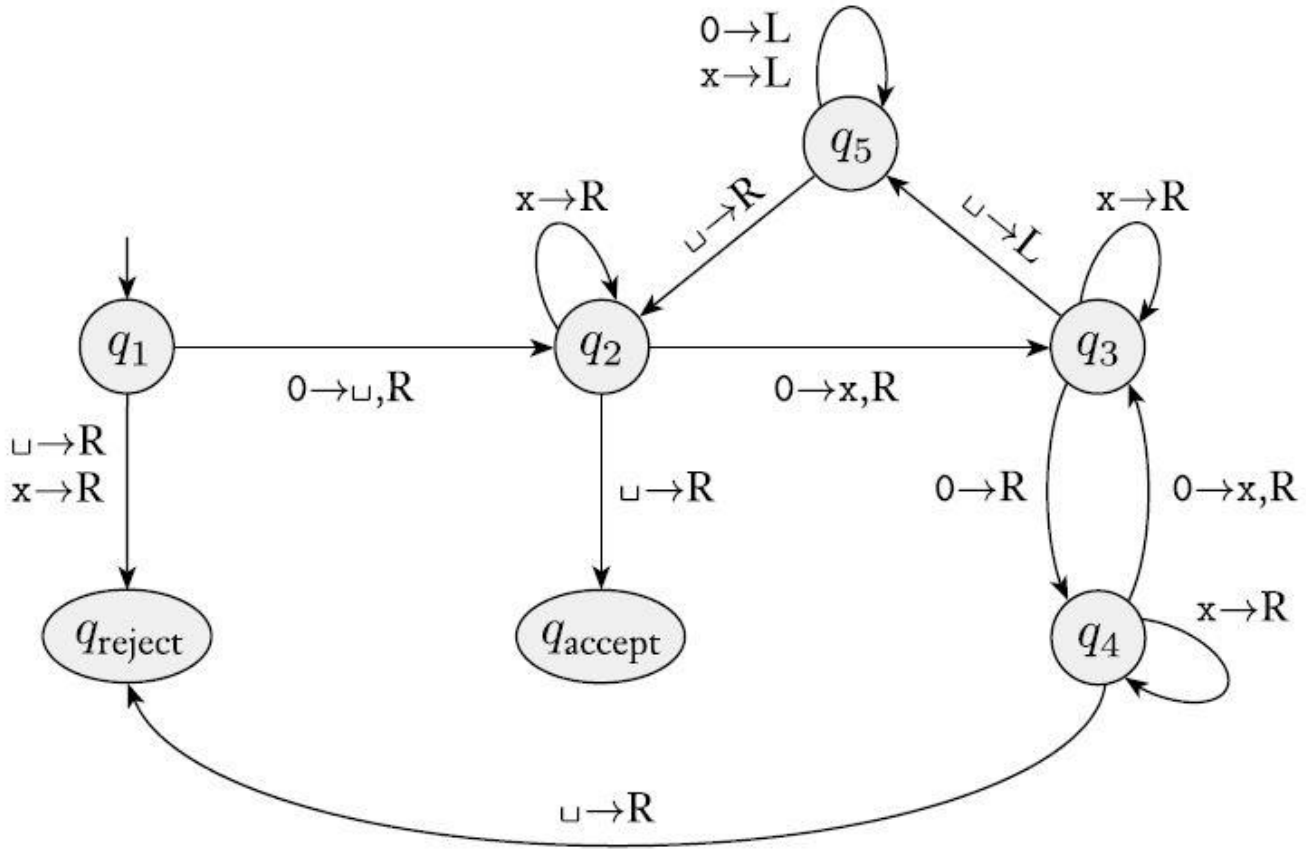
- Q is a finite set called states,
- Σ is a finite set called the input alphabet, not containing the symbol u
- Γ is a finite set called the tape alphabet, where $u \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_{acc} \in Q$ is the accept state and $q_{rej} \in Q$ is the reject state, where $q_{acc} \neq q_{rej}$

Example

- Consider the language $L = \{ 0^{2^n} \mid n \geq 0 \}$

Example

- Consider the language $L = \{ 0^{2^n} \mid n \geq 0 \}$

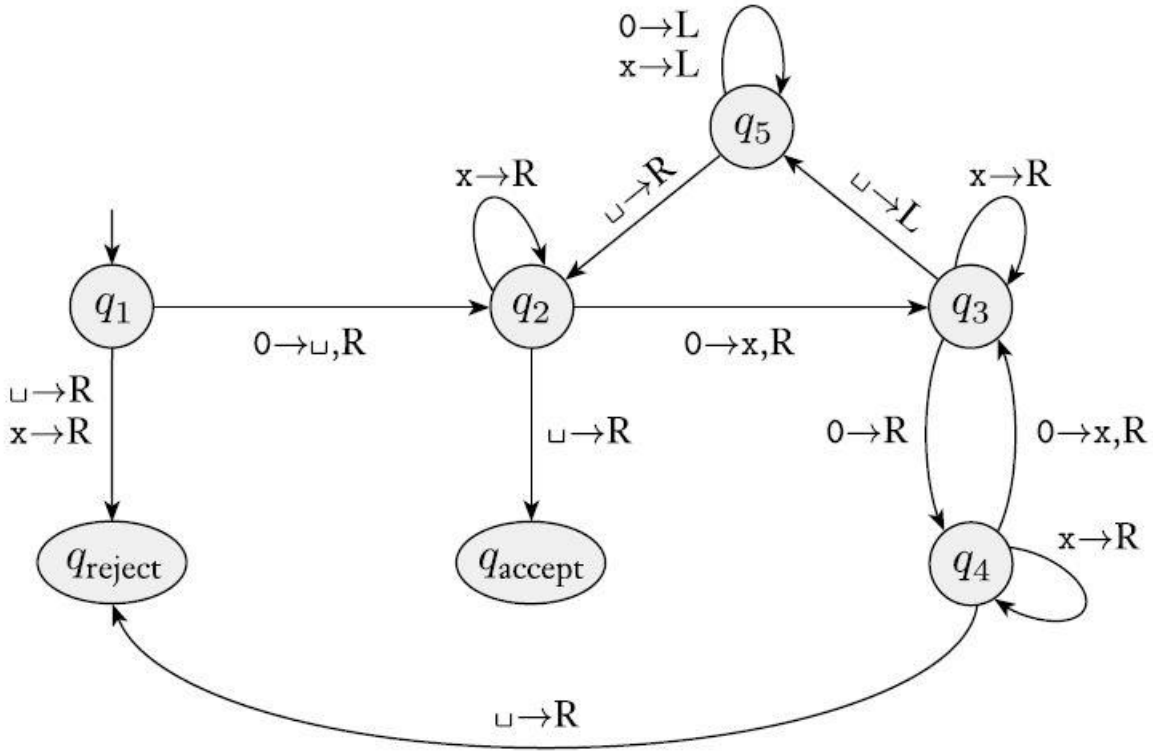


Example

- Consider the language $L = \{ 0^{2^n} \mid n \geq 0 \}$

On input string w:

1. If input is invalid, reject.
2. Sweep right across the tape, cross off every other 0 (one in every two).
3. If in stage 2 the tape contained a single 0, accept.
4. If in stage 2 the tape contained more than a single 0 and the number of 0s was odd (even, ignoring the first one), reject.
5. Return the head to the left-hand end of the tape.
6. Go to stage 2.



Instantaneous Description (Configuration)

- As a Turing machine computes, changes occur in the current state, the current tape contents, and the current head location. A setting of these three items is called a configuration of the Turing machine.

- For a state q and two strings u and v over the tape alphabet Γ, we write u q v for the configuration where the current state is q, the current tape contents is uv, and the current head location is the first symbol of v.

Instantaneous Description (Configuration)

- Say that configuration C1 yields ($\vdash$) configuration C2 if the Turing machine can

  legally go from C1 to C2 in a single step.

- For example, Say that

$$ua\ q_i\ bv \quad \vdash \quad u\ q_j\ acv$$

  when $\delta(q_i, b) = (q_j, c, L)$.

- $\vdash^*$ can be defined accordingly for zero or more steps.

Instantaneous Description (Configuration)

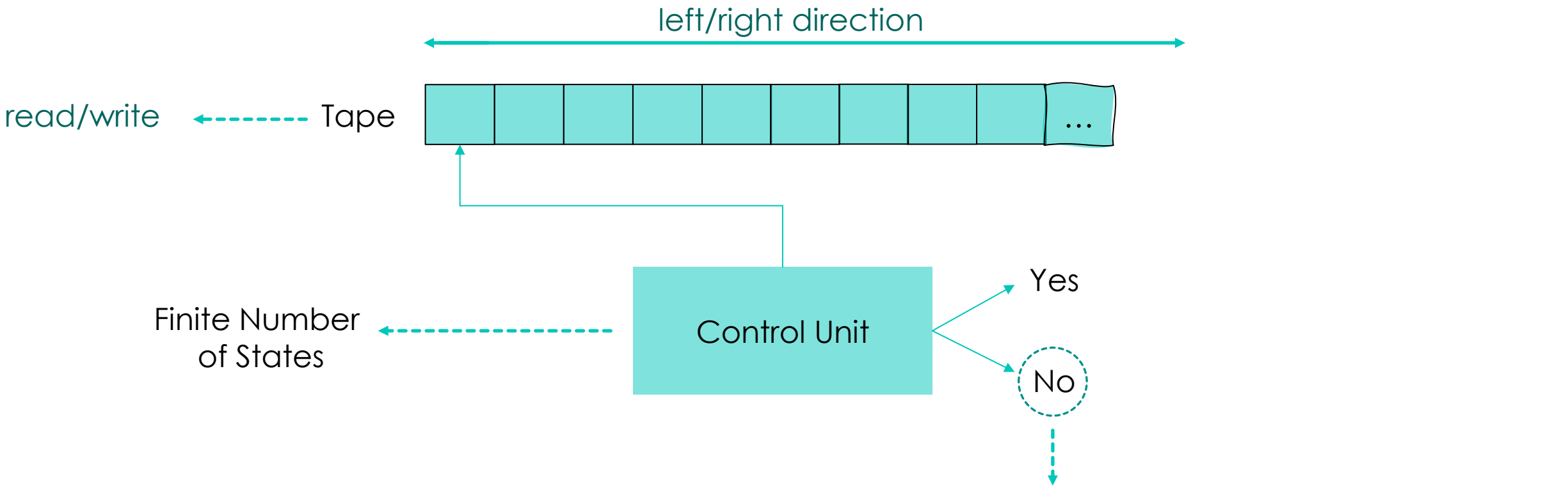
For any legal Turing Machine M and any string $w \in \Sigma^*$ and $u, v \in \Gamma^*$:

- The start configuration of M on input w is the configuration $q_0 w$
- In an accepting configuration, the state of the configuration is $u\ q_{acc}\ v$
- In a rejecting configuration, the state of the configuration is $u\ q_{rej}\ v$
- Accepting and rejecting configurations are halting configurations and do not yield further configurations.

String and Language Recognition

- A Turing machine M recognizes (accepts) input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exists, where
  1. C_1 is the start configuration of M on input w, (i.e. $C_1 : q_0w$)
  2. each C_i yields C_{i+1}, and
  3. C_k is an accepting configuration.
- A Turing Machine M recognizes (accepts) a language L if for every $w \in L$, M halts in an accepting state, and for every $w \notin L$, M either rejects or never halts.

Turing Machine (as an acceptor)

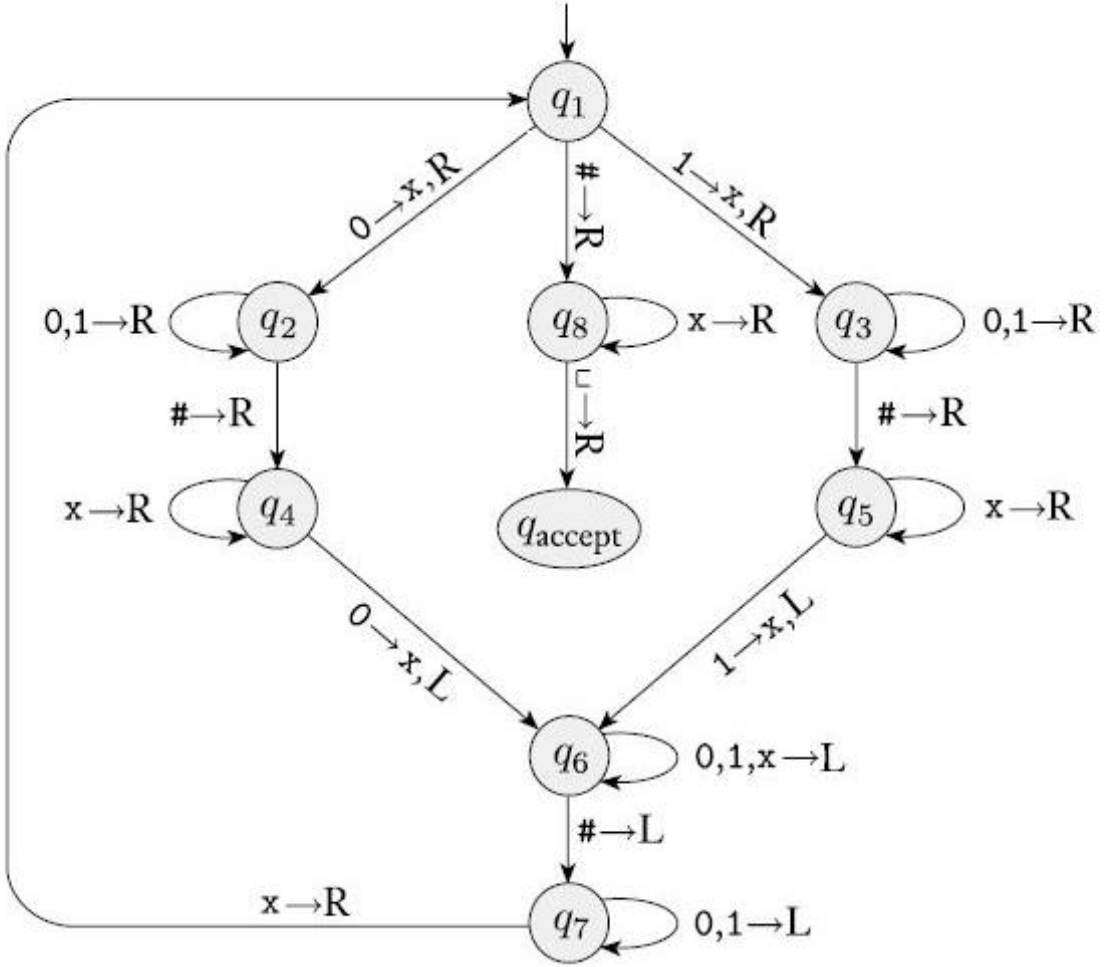


Example

- Consider the language $L = \{ w\#w \mid w \in \{0, 1\}^* \}$

Example

- Consider the language L = { w#w | w ∈ {0, 1}*}



the reject state is not depicted here in the picture for simplicity, and we assume that all absent transitions go to this state implicitly.

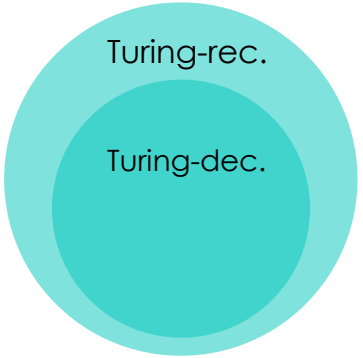
Turing-recognizable/-decidable Languages

- The set of all languages for which there exist a Deterministic Turing Machine that recognizes their strings, is called the class of Turing-recognizable languages (also called the class of recursively enumerable languages).

- The set of all languages for which there exist a Deterministic Turing Machine that recognizes their strings and always halts on every input, is called the class of Turing-decidable languages (also called the class of recursive languages).

Turing-recognizable/-decidable Languages

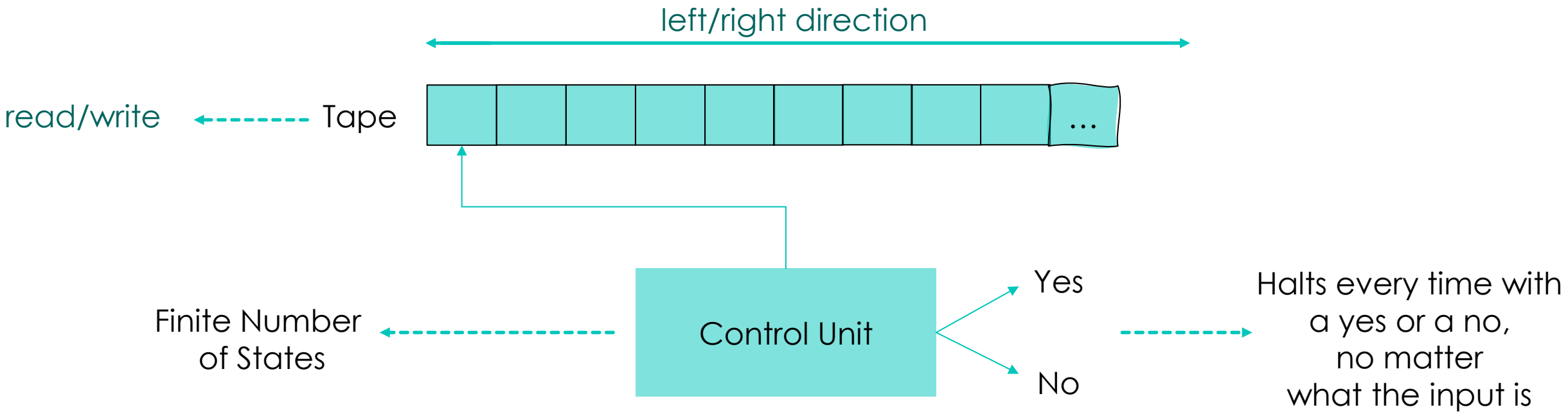
- The set of all languages for which there exist a Deterministic Turing Machine that recognizes their strings, is called the class of Turing-recognizable languages (also called the class of recursively enumerable languages).



We call these “Decider Turing Machines”

- The set of all languages for which there exist a Deterministic Turing Machine that recognizes their strings and always halts on every input, is called the class of Turing-decidable languages (also called the class of recursive languages).

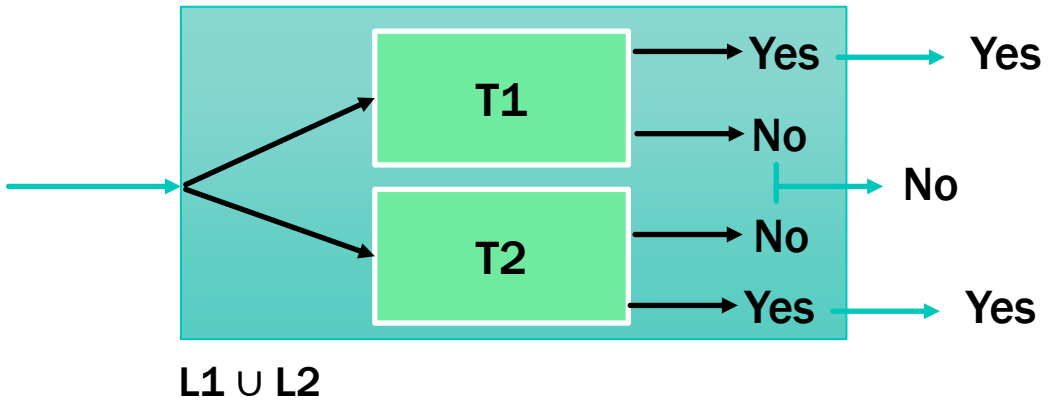
Decider Turing Machine (as an acceptor)



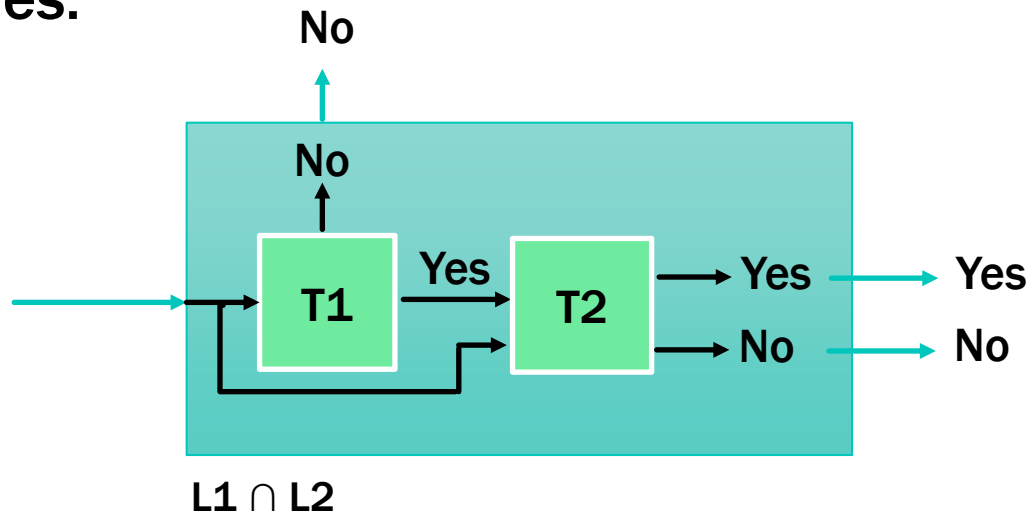
Operations on TR Languages : Union and Intersection

- **Theorem**: If L_1 and L_2 are both Turing-recognizable languages over the same alphabet Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also Turing-recognizable.
- **Proof**: Suppose that T_1 is a TM that accepts L_1, and T_2 is a TM that accepts L_2.

Now consider the following Turing machines:



$L_1 \cup L_2$



$L_1 \cap L_2$

Question

- For the sake of assumption, consider the class of Turing-recognizable languages being closed under the complement operator. What is the immediate conclusion?

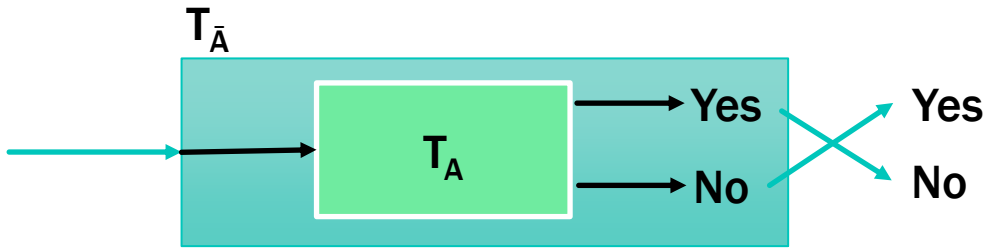
- Try to prove or disprove that the class of Turing-recognizable languages is closed under the Kleene star, concatenation and complement operations.

- Do the same with the class of Turing-decidable languages, considering also union and intersection operations.

Operations on TD Languages: Complement

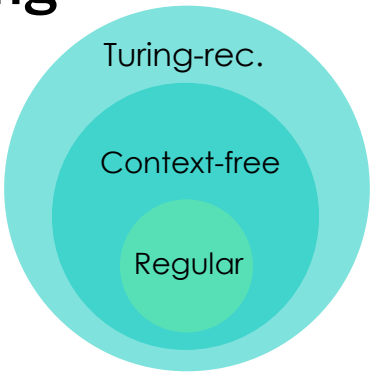
- Because of its utmost importance, we will ourselves answer the question regarding the closure of Turing-decidable languages under the complement operator.

- Consider T_A to be a decider TM for language A. The TM $T_{\bar{A}}$ which decides the complement language $\bar{A}$ can be constructed as follows:

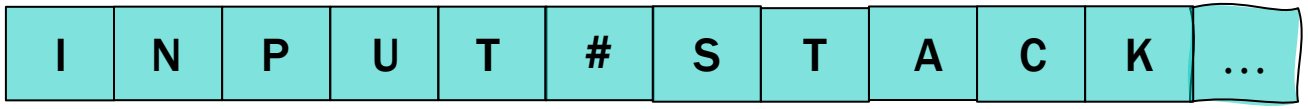


Relation with Context-free Languages

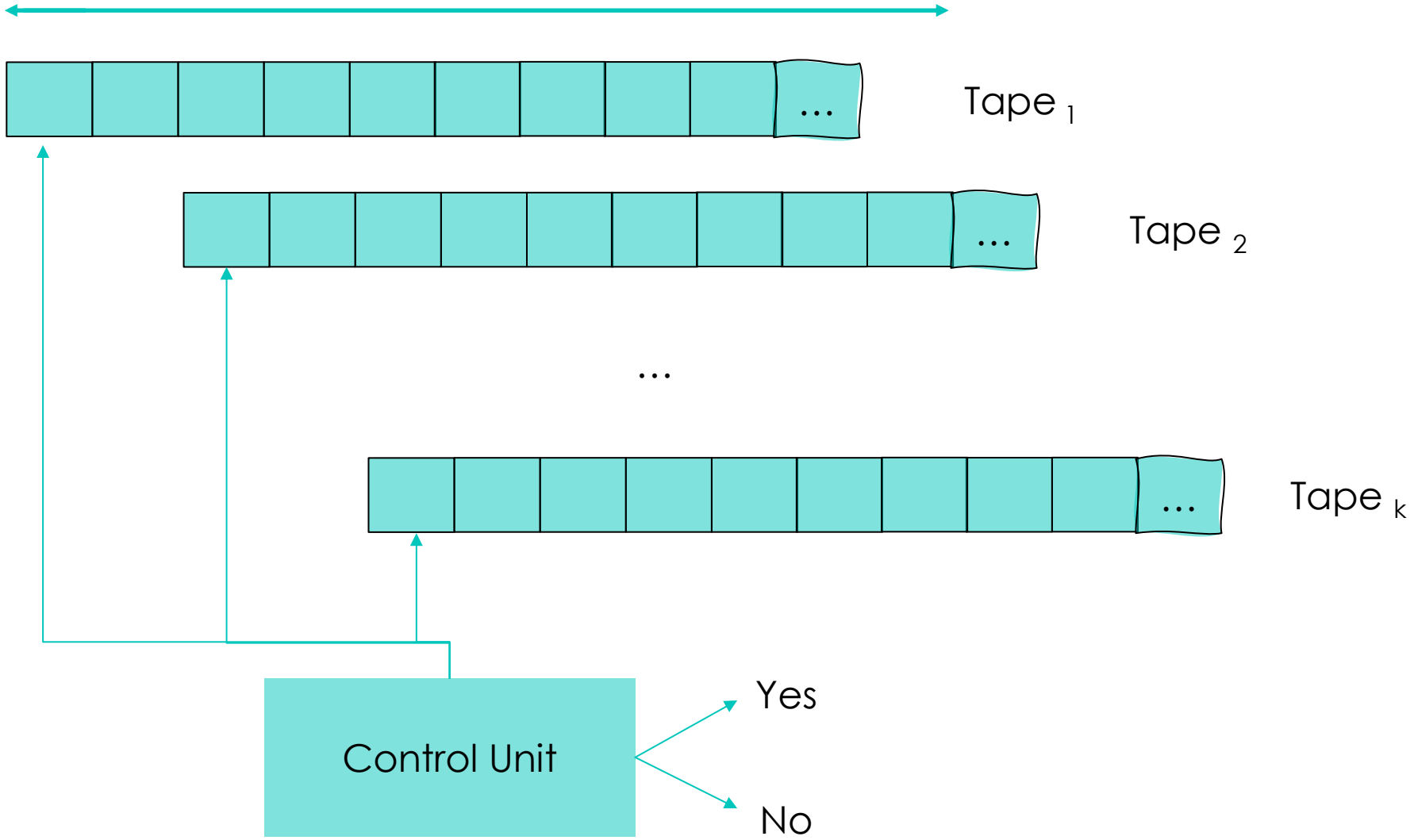
- The class of context-free languages is a proper subset of the class of Turing-recognizable languages.



- In order to simulate a PDA as a TM, we separate the tape into two parts, one for the input and one for the stack, and as the TM can move forward and backward through the tape, this is sufficient for the implemention.



Multi-tape Deterministic Turing Machines



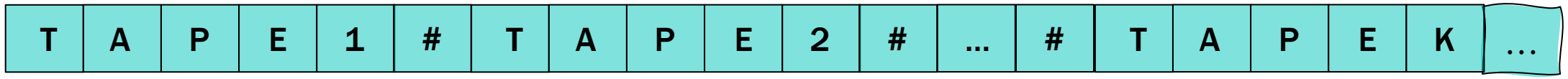
Multi-tape Deterministic Turing Machines + STAY

A Deterministic Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set called states,
- Σ is a finite set called the input alphabet, not containing the symbol u
- Γ is a finite set called the tape alphabet, where $u \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma^k \longrightarrow Q \times \Gamma^k \times \{L, R, S\}^k$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_{acc} \in Q$ is the accept state and $q_{rej} \in Q$ is the reject state, where $q_{acc} \neq q_{rej}$

Multi-tape Deterministic Turing Machines

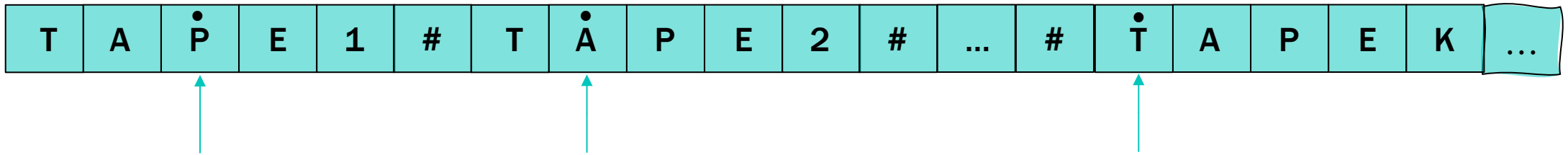
- Every multi-tape Turing Machine M has an equivalent single-tape Turing machine S. To simulate a multi-tape TM, we use the same idea as before:



- The only question that remains is how to keep track of the heads of the tapes?

Multi-tape Deterministic Turing Machines

- Every multi-tape Turing Machine M has an equivalent single-tape Turing machine S. To simulate a multi-tape TM, we use the same idea as before:



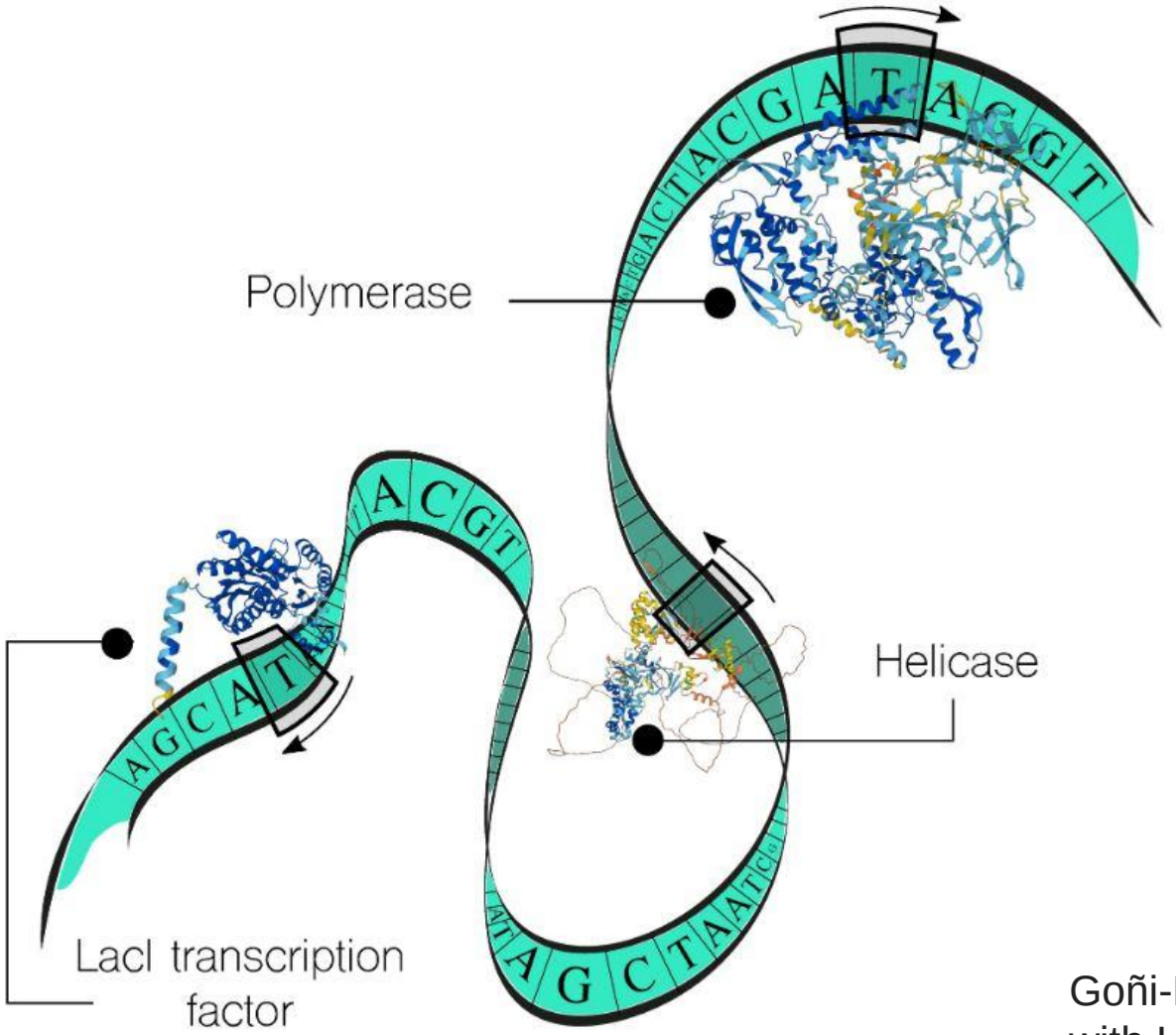
- We use special characters that each corresponds to one ordinary character in the tape alphabet, to mark the place where the head on that specific tape would be.

Example: DNA

- Imagine DNA as a Turing Machine (TM). In this analogy, DNA is like the infinite tape in a TM with symbols written in it. In the case of DNA, the input alphabet is limited to four symbols: A, C, G, and T.

- The system can be modeled as a multi-tape Turing machine, with each enzyme slides along a unique tape in both directions, reading, writing, and performing other tasks.

Example: DNA



Goñi-Moreno, Á. (2024). Biocomputation: Moving Beyond Turing with Living Cellular Computers. *Communications of the ACM*.

Example: DNA

- This system is not yet fully understood, nor explicitly formalized as a computing device, but it offers exciting possibilities for unlocking the true information-processing power of living systems.

- One example is conceptualization of such an enzymatic apparatus as a computing system capable of executing logically reversible functions, an approach that has recently undergone experimental analysis.

Non-deterministic Turing Machines (NTM)

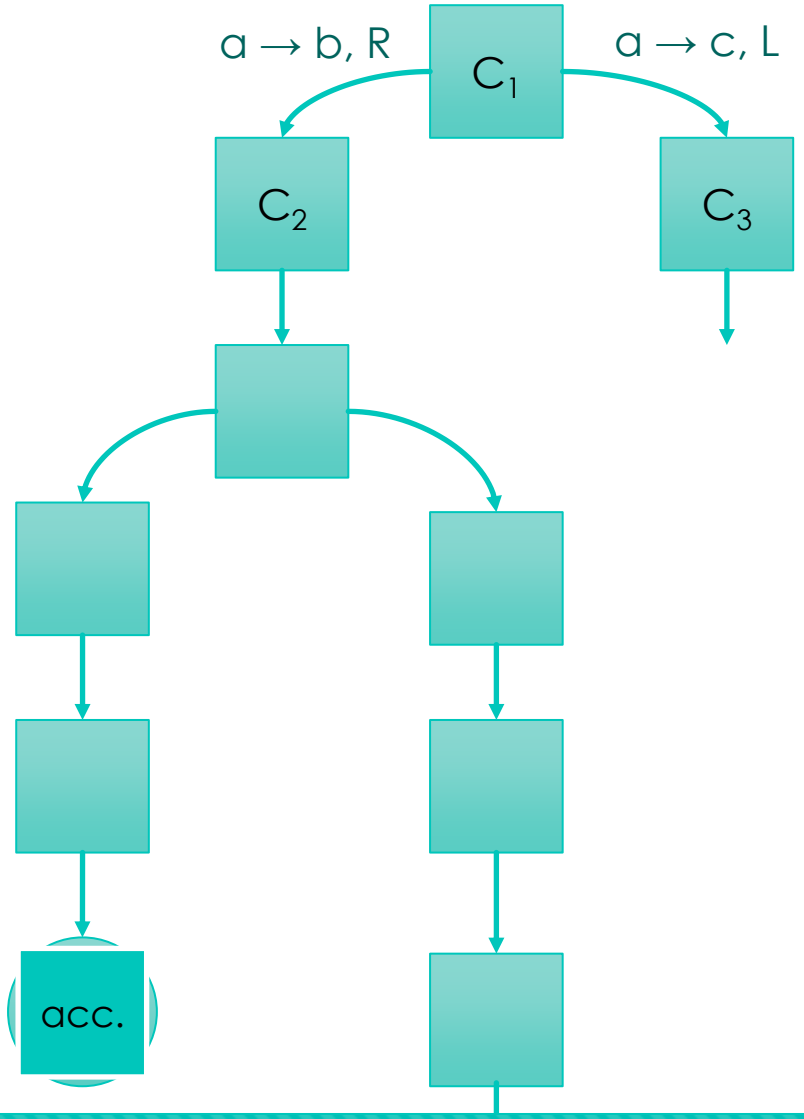
A Non-deterministic Turing Machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ where

- Q is a finite set called states,
- Σ is a finite set called the input alphabet, not containing the symbol u
- Γ is a finite set called the tape alphabet, where $u \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \longrightarrow 2^{Q \times \Gamma \times \{L, R\}}$ is the transition function,
- $q_0 \in Q$ is the start state,
- $q_{acc} \in Q$ is the accept state and $q_{rej} \in Q$ is the reject state, where $q_{acc} \neq q_{rej}$

Non-deterministic Turing Machines (NTM)

- Every non-deterministic Turing machine N has an equivalent deterministic Turing machine D.

- To simulate a non-deterministic TM, we consider the following tree where each node represents a configuration in the NTM and each branch represents an alternative computation path.

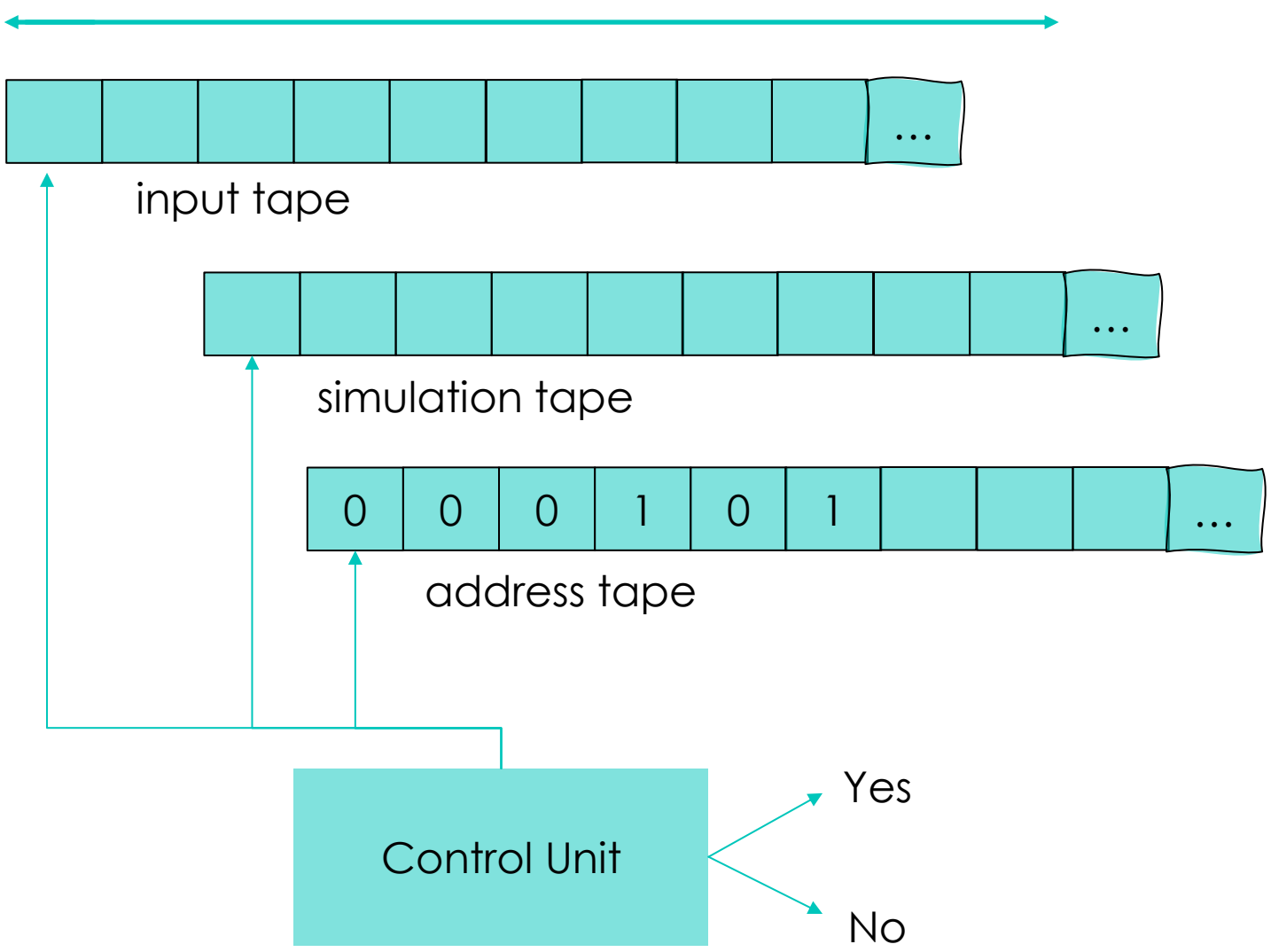


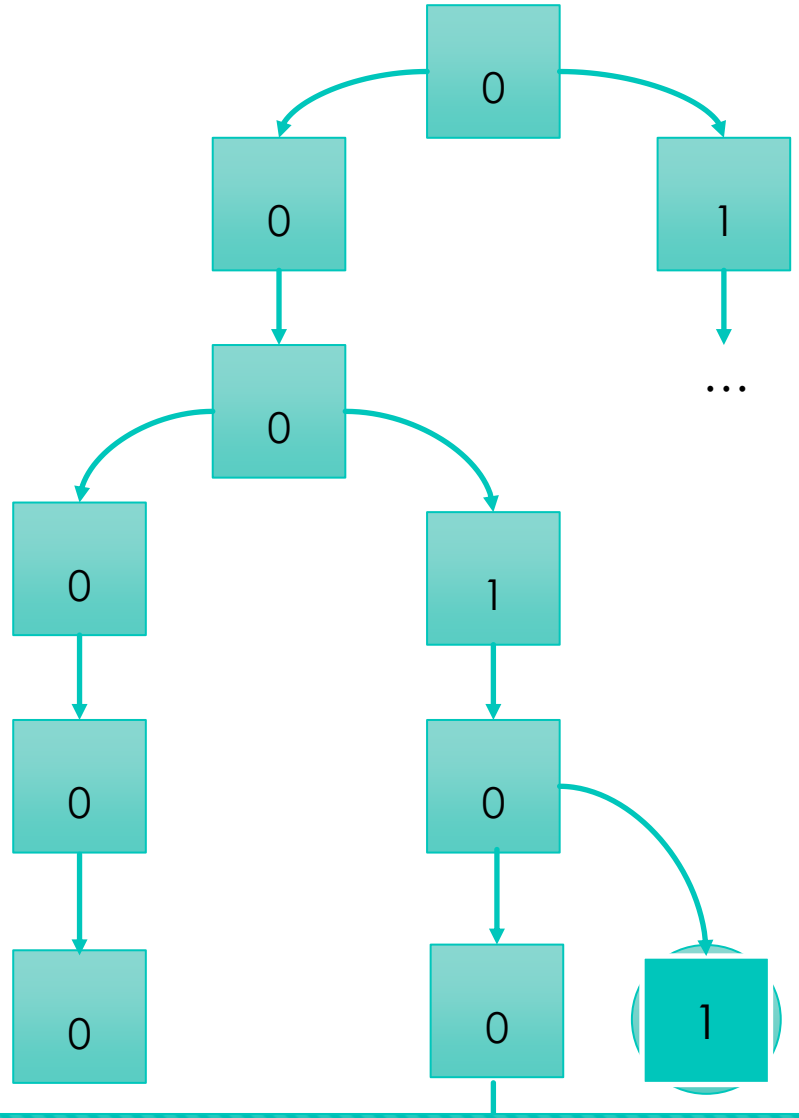
Non-deterministic Turing Machines (NTM)

- We then build a 3-tape deterministic turing machine that traverses the tree in a breadth-first manner, looking for an accept configuration.

1. The first tape contains the input string, which is never altered (input tape).
2. The second tape maintains a copy of tape on some branch (simulation tape).
3. The third tape keeps track of the current node in the computation tree (address tape).

Non-deterministic Turing Machines (NTM)





Function Problems vs. Decision Problems

- Function Problems are computational problems that require a single output (of a partial function) for every given legal input.

- The problems in this class can be formulated as:

Given a function f and an input $x \in dom(f)$, compute $f(x)$

Function Problems vs. Decision Problems

- Each such function problem can be represented by the following set:

$$L = \{ (x, y) \mid x \in \text{dom}(f) \text{ and } f(x) = y \}$$

- We can also describe certain other characteristics of the function problems by representing them as languages, such as the following:

$$L = \{ x \mid \text{there exist y such that } f(x) = y \}$$

Function Problems vs. Decision Problems

- Decision Problems are computational problems that require a single yes/no output for every given legal input.

- Every membership problem regarding a language can be categorized as a decision problem. We can conclude that for every function problem, there are corresponding decision problems that ask for certain characteristics in the graph of its partial function.

Hilbert's Tenth Problem

- In 1900, mathematician David Hilbert delivered a now-famous address at the
- International Congress of Mathematicians in Paris.

- In his lecture, he identified 23 mathematical problems and posed them as a challenge for the coming century. The tenth problem on his list is of importance here.

Hilbert's Tenth Problem

- Hilbert's tenth problem was to devise an algorithm that tests whether a polynomial has an integral root or not.

- Interestingly, in the way he phrased this problem, Hilbert explicitly asked that an algorithm be "devised." Thus he apparently assumed that such an algorithm must exist and someone need only find it.

Hilbert's Tenth Problem

- The following decision problem can be formulated based on Hilbert's tenth problem.

- Consider the following language:

$$D = \{ p \mid p \text{ is a polynomial with an integral root} \}$$

The problem asks, Given the language D and polynomial x, whether $x \in D$ or not

- The problem is essentially asking whether a decider Turing machine exists for D.

Hilbert's Tenth Problem

- We now know that this problem is undecidable, i.e. we can not define a decider Turing machine that given any polynomial, decides whether it has an integral root or not. Note that there indeed exists a non-decider TM M that recognizes the language, the definition follows:

On input p,

if p isn't a polynomial, reject,

else evaluate p with setting the variables successively to the values 0, 1, -1, 2, ...

and if at any point, the polynomial evaluates to 0, accept.

The Church-Turing Thesis

- For any membership problem regarding a language L :

1. There exists an algorithm (in an intuitive philosophical concept) to compute (decide, yes or no), whether a given string belongs to the set L, if and only if L is a Turing-decidable language.

2. There exists a semi (or partial) algorithm if and only if L is a Turing-recognizable language.

   Here, the partial algorithm means a procedure that gives an answer of yes for exactly the strings in L, but may fail to give an answer for strings that are not in L.

Universal Turing Machines

- The Turing machines we have studied so far have been special-purpose computers, capable of executing a single [partial] algorithm.

- Just as a modern computer is a stored-program computer, which can execute any program stored in its memory, so a Universal Turing Machine, introduced by Turing in a 1936 paper, can execute any algorithm, provided it receives an input string that describes the algorithm and any data it is to process.

Universal Turing Machines

A Universal Turing Machine is a Turing machine T_u that works as follows. It is assumed to receive an input string of the form e(T)e(z), where :

- T is an arbitrary Turing machine,
- z is an arbitrary string over the input alphabet of T
- e is an encoding function whose values are strings in $\{0, 1\}^*$
- The [partial] computation performed by T_u on this input string satisfies the following property: T_u accepts the string e(T)e(z) if and only if T accepts z

Encoding a Turing Machine

Let T be the set of all Turing machines. We can a define a one-to-one function

$e : T \rightarrow \{0, 1\}^*$ in the following way.

Consider TM = $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$ is a Turing machine. We define the string e(TM):

1. We assign unique numbers to each state, tape symbol, and tape head direction.
2. For each move m of the form $\delta(p, z) = (q, v, D)$ where $p, q \in Q$, $z, v \in \Gamma$, $D \in \{R, L\}$ we define

$$e(m) = 1^{n(p)}01^{n(z)}01^{n(q)}01^{n(v)}01^{n(D)}0$$

3. We list the moves of TM in some order $M_1, M_2, \ldots, M_k$ and we define

$$e(TM) = e(M_1)0e(M_2)0e(M_3)0\ldots0e(M_k)0$$

Notation

From this point on, for simplicity, we will use <T> instead of e(T) to show the encoding of a Turing Machine or any other objects, and when multiple objects O_1, ..., O_k are involved, we denote their encodings into a single string $\langle O_1, ..., O_k \rangle$

Theorem 1

Theorem: The set of all Turing Machines is countable.

Proof: In order to prove the above statement, we first need to note that a Turing machine TM can be represented by the string <TM>, and each such string can represent at most one Turing Machine. Therefore, we may think of the function as a bijection from T to a subset of $\{0, 1\}^*$. Because $\{0, 1\}^* = \bigcup_{i=1}^{\infty} \{0, 1\}^i$ is countable (it's a countable union of countable sets), every subset is, and we can conclude that T is countable.

Two Corollaries

- Because every Turing machine represents a single language, we immediately conclude that:

  Corollary 1: The set of Turing-recognizable languages is countable.

- Because every subset of a countable set is also countable, we also conclude that:

  Corollary 2: the set of Turing-decidable languages is countable.

Theorem 2

Theorem: There exist languages that are not Turing-recognizable.

Proof: We know that the set of subsets of $\mathbb{N}$ is uncountable. Because $\{0, 1\}^*$ is the same size as $\mathbb{N}$, it follows that the set of all languages over $\{0, 1\}^*$ is uncountable. We know (previous slide) that the set of all Turing-recognizable languages is countable. Because when T is any countable subset of an uncountable set S, S − T is uncountable, we conclude that there are uncountably many languages that are not Turing-recognizable.

